GROUP 28
MUSIC GENERATION WITH NEURAL COMPOSER

Lynsey Johnson, Sushmitha Kudari, and Avinash Mallavarapu

University of California San Diego, La Jolla, CA 92093-0238,

ABSTRACT

Since the beginning of great composers, music has always been a painstaking process to write and compose. This process no longer needs to be restricted to manual labor done by humans, but instead can be automated using AI. HackerPoet has built a neural composer that is trained on a MIDI dataset of video game theme songs and outputs an artificially generated game theme song. The architecture is composed of an encoder and a decoder which take in Midi files and generate piano roll format from which the model learns. While the idea of a neural composer is not a new one, we aim to modernize and generalize HackerPoet's Composer. The code that HackerPoet has presented is deprecated and archaic. We have modified the existing repository to be more robust and user friendly. Through this replication and restoration process we were able to generate tonal sounds that somewhat resemble modern day abstract music.

***Index Terms*—** autoencoder, generative adversarial network, GAN, neural network, note complexity, music generation

1. INTRODUCTION

A neural composer would remove the need for the painstaking work of composing snippets of music for different scenarios. It serves as an optimizer in both memory and speed for small movie scores, video games and sound effects. This would allow video games and movie scores to be generated on the fly without needing human interference.

HackerPoet has built a neural composer that is trained on a MIDI dataset of video game theme songs. Currently, the composer converts all training data into piano roll format and treats every note as a single strike with no holds, decomposing 96 note samples into time steps of 96 units per measure and adding a third dimension of 16 measures to capture the symmetry of repeated measures and motifs in highly structured songs. Each measure is encoded into a feature vector, which then is fed into an autoencoder which produces another feature vector for the entire song, then is finally run through decoders to be converted back into independent measures. The baseline composer network is essentially two identical encoders with opposite feed directions. The hyperparameters for each measure encoder are then adjusted to improve the results. The model for the Neural Composer itself is not a complete one. While it sounds good in the videos seen online, the composer is not robust enough to handle piano roll music and therefore needs modifications and upgrading. After our adaptations to this Neural Composer, the model takes no input since this is a Generative Adversarial Network. The output is a series of notes put together to produce a "semi-fluid" song.

2. RELATED WORKS

The idea of artificial music generation came about in 1988. The first GAN was a mulit-layer perceptron that composed music on the principles of "creation by refinement." This model used gradients to learn "musicality." As the model took a gradient walk, it would select the tones that sounded the most pleasant. Peter M. Todd then built on this feed-forward network by using a regressive neural network (RNN) to generate music sequentially in 1989[**?**]. He used a windowed method of processing successive time-periods of melodies similar to speech applications at the time. His model was reusing the results of the prior calculations and learning temporal sequences.

Douglas Eck and Jurgen Schmidhuber built on the recurrent network model with Long short-term memory (LSTM) structure in 2002 [4]. LSTM is commonly used in language generation to teach models to remember prior contexts of long sentences so the whole structure of the sentence is understood. LSTM networks utilize a cell state that windows through processing gates that measures and recognizes similarities throughout different segments of a large piece of data (e.g a long sentence or a song). LSTM models are designed to learn long term dependencies and patterns making it especially useful to composition applications. Simply put, they were using self attention to remember portions of songs that were heard previously and use what is being learnt in the current context to generate music in the same structure.

3. DATASET AND FEATURES

The dataset we used was generated and compiled by Colin Raffel in "Learning-Based Methods for Comparing Sequences, with Application to Audio-to-MIDI Alignment and Matching". Raffel generated the dataset by developing a series of learning-based methods to compare, identify and match entries in the Million Song Dataset. These entries are then converted to audio files and to MIDI format. Researchers at the Music and AI Lab derived labels for the genres of audio files contained in the Lakh MIDI Dataset based on their mapping to the Mission Song Dataset and converted them to piano roll format. The piano roll data divides the MIDI file into five different separate tracks including drums, piano, guitar, bass and strings. Our group wrote a script to pull all the labels identified as Jazz from the dataset, removing duplicates from different sources (lastfm, tagtraum, etc), ending with about 2500 Jazz labeled MIDI songs in total.

3.1. Data Preprocessing

Since the dataset contained real jazz songs from a variety of sources, we needed to normalize the dataset for training. We first read in all 2500 MIDI songs identified as jazz and converted them to numpy arrays for ease of use in python using the mido library. The input MIDI songs were mixed between octave intervals and a hundred twenty-eight note, or quasihemidemisemiquaver intervals. For the latter, we converted those to octave intervals as a first iteration of normalization. Then the samples were then divided into measures using the equation

$$t_m = t_{abs} * n/t_x. \quad (1)$$

Where t_m is equal to the number of beats in a measure, t_{abs} is the absolute time relative to the start of the track, n is the number of samples per measure equal, and t_x is the number of ticks per measure in the song. We then added zero-padding around each measure of the size [n x n] for the model to more easily identify and discern measure structure from in training. Lastly, we binarized the notes to make the songs more similar to a uniform time series type data input and cropped the note arrays to sixteen measures centered around the midpoint of each song.

4. METHODS

We begin by decomposing single note samples into singular time steps. Each song is represented as a 3 dimensional array of size 16x96x96. Each measure (size 96x96) is then passed through 16 different fully connected layers to generate feature vectors having sizes of 200 which represent the input measures. These 16 feature vectors are concatenated and passed through another encoder to generate a song feature vector. The bottleneck layer has a size of 120. Once we have the latent representation of the song, we pass it through a dense layer to get 16 feature vectors. These feature vectors are then propagated through a second decoder to convert back into independent measures. The loss function is a combination of reconstruction loss and latent (VAE) loss. The reconstruction (L2) loss is used to compare the generated song and input song. L2 loss tries to predict the the mean of the distribution to minimize the MSE. This can lead to distortion in the output signal. To alleviate this issue, we add a latent loss. Latent loss is the same as the KL divergence loss which penalizes the model if the latent vector (dim = 120) doesn't come from a gaussian distribution with zero mean and identity covariance matrix. The Neural composer was then trained for 500 epochs at a learning rate of 0.001 with batch size 350. RMSprop optimizer was used to train the model.

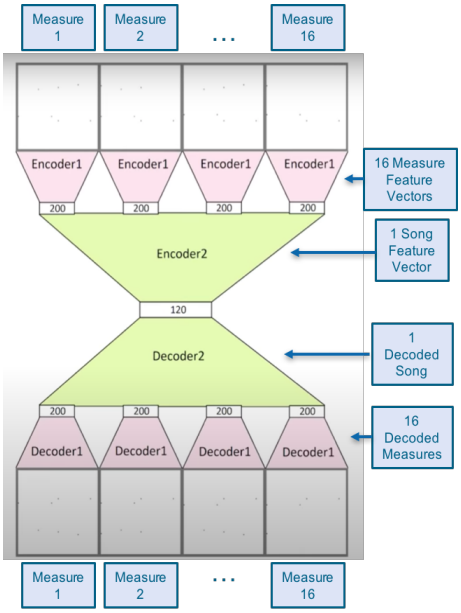

Fig. 1. Variational Autoencoder Model

Convolutional GANs have shown promising results in modeling image generation. Inspired from MidiNet and HackerPoet, we develop a convolutional GAN to generate music. The generator network consists of 4 transposed convolutional layers which help in transforming the input noise vector to a song vector of size 16x96x96. Each transposed convolution is followed by batch normalization and LeakyRelu activation. The discriminator network consists of 4 convolutional layers. Instead of using pooling layers, we decided to use strided convolutions. After applying the 4 convolutional layers, we finally apply a sigmoid activation to get an output in the range [0,1] (determines the probability of a song being real or fake).

5. EXPERIMENTS/RESULTS/DISCUSSION

A large part of our experiment in this project was centered around the configuration of the data as input to the autoencoder model. Using audio files was a new realm for all individuals in the group and proved to have a steep learning curve associated with working with music samples as training data. The characteristics of the domain of music as a whole was a challenge in itself as we had to learn metrics for normalizing our data based on song sampling, tempo, note frequency, progression and measure structure. Additionally a larger chunk of time than anticipated was spent getting the reference variational autoencoder model working as it was using unsupported keras and tensorflow versions. The process of modernizing the network and all the dependencies was well above what we estimated as a group in our project plan.

For our functional music generation experiments, we trained the variational autoencoder and evaluated results iteratively through 500 epochs of training. At each evaluation step we recorded the loss and submitted two audio samples to the decoder/predictor to test the models progression. The first sample sent to be tested on the decoder was a real song pulled from the dataset in the same preprocessing format to those it was trained on. The second sample submitted to the decoder was an array of random noise shaped in the same dimensions as the preprocessed data. The purpose of this experiment was to determine how developed the feature vector trained by the autoencoder portion of the model was, and how much it would change either sample.

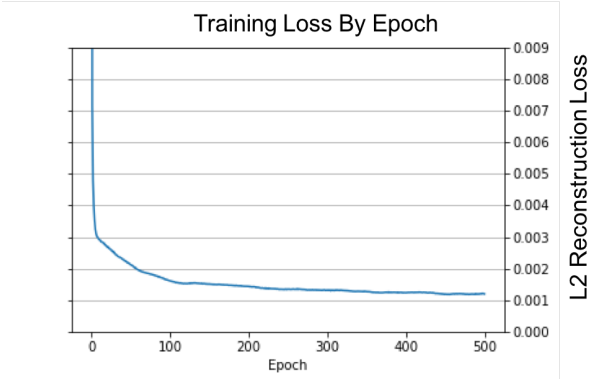

Fig. 2. Variational Autoencoder Training Loss by Epoch

Figure 3, shows the predicted song audio complexity using a true song input as input. As epochs increase, the generated song starts to more closely resemble the note complexity of the input song. This shows that the model did in fact learn patterns of chord progression stylistic qualities seen in the jazz dataset. Instead of diverging and adding more variety of musical notes, the decoder uses the feature vector to keep consistent yet built upon the input song. The produced song in the end still looks very much like a real song.

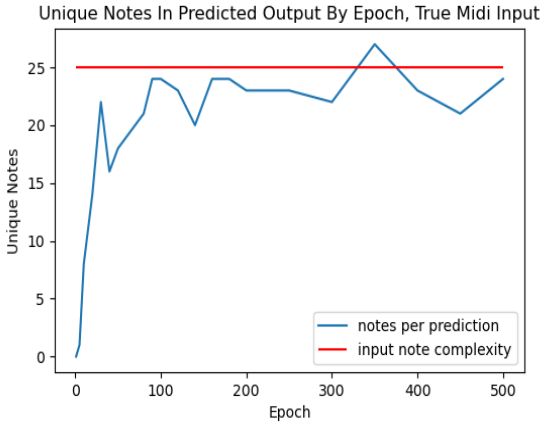

Fig. 3. Variational Autoencoder Prediction Results on Real Data

Alternatively, figure 4 presents the decoded song audio complexity using a random noise array reshaped to look like the input data. The note complexity of the random noise vector is much higher than that of the real data input discussed in the previous experiment portion as it does not follow any stylistic note grouping or music chord progression. The model struggles refining the chaotic noise input to something that looks more song-like using the trained feature vector in early epochs, but as they increase the model is successful in refining the song to the level of complexity and uniformity of the training data. This tells us that the model has indeed learned the jazz stylings enough to refine a random noise input to appear to be a unique song with similar note complexity to the training data. The average note complexity of the training set is about thirty notes per song, which is close to where the model converges as epochs increase.

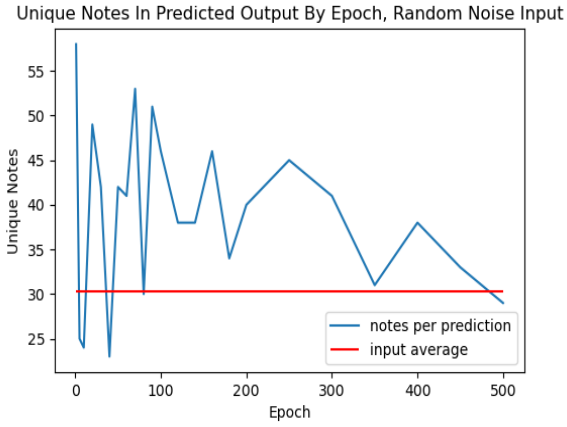

Fig. 4. Variational Autoencoder Prediction Results on Random Noise Data

The complete code to our data preprocessing, random noise generation, experiments, results compilation, and attempt and modernizing other models can be found in the GitHub repo linked below.

$https://github.com/eroooon/ECE228_Project$

6. CONCLUSION

Reconstructing and updating HackerPoet's Neural composer was much more complex than intended. We discovered that using music as a datasource is much different than image processing. There is a new layer of complexity involved because of the element of temporal frames. Overall, our reconstructed model trained well on the data it was given after it went through normalization and preprocessing. A new discovery we made was how representative the output of GANs are of their training datasets. This concept most likely translates over to GAN images as well. We do wish that we had more resources such as better audio data sets and lesser complex models than we intended to build so that we could have a point of comparison using the same training dataset. Overall it is very evident to see that unlike classifiers and most other networks, GANs are very unique in their ability to replicate human behavior and may soon be surpassing humans in the world of art.

7. CONTRIBUTIONS

7.1. Lynsey Johnson

Primarily MIDI dfdata preprocessing, pianoroll formatting as binary temporal tonal representations of various instrument notes, and preparing the isolated bass, drums, guitar, piano and string tracks to learn. Due to processing and resource limitations, Lynsey performed the training experiments and compiled the results. Lynsey also attempted to implement a MidiNet Composer to compare it to Hacker Poet's Composer.

7.2. Sush Kudari

Implemented the Hacker Poet's Neural Composer to use as a comparison composer. Since the output is generated music and does not fall under classification, she looked into how to compare the output of the final song's complexity to the other songs generated during the model's training. Wrote helper scripts to support the other two members.

7.3. Avinash Reddy

Attempted to implement the progressive training technique on Neural Composer which would help stabilize the training of the GAN. He performed experiments by making tweaks in the architectures used in Neural Composer (for eg:- using transposed convolution in place of max unpool in the generator network)

8. REPLY TO REVIEWS

8.1. Group 15

The project is really interesting and their goal is pretty clear.Strong Explanation on neural composer model (Variational AE model). Great presentation from all. It would have been better to see the presenter's cam.Is there any reason that you used Jazz MIDI for the neural composer model and the Jazz piano roll data for the progressive MidiNet model? How about using both models for both data and comparing the result?Did you use the MiDiNet and Convolutional Gan model? There are no results for them? We did not use a Convolutional Gan. Out neural composer is an autoencoder model.

Response: We attempted to implement the MiDiNet model but the code is very legacy.

Would have been nice to have a loss plot for both training and test and lasso for all 3 models that you used.

Response: Since this is a GAN we do not have a testing phase in our model.

8.2. Group 24

The presentation and demo were well organized and clear. They gave an in-depth explanation on why creating a music generator is important and what research has been done. Comparing the performance of the MidNet and the Neural Composer Model is a very interesting idea because these two models have totally different architectures; one used a generator and discriminator whereas the other used an autoencoder. The talk also explained the dataset, network architecture of the MidiNet and the Neural Composer Model and presented visualization on the results. The "Proof of Increasing Complexity over Epochs" shows that the model is able to learn and create more complicated outputs. For the demo at the end, the output music file sounds like it only has one note and what might be the reason?

Response: There are actually multiple notes present in this song. The output just doesn't sound like our expected "music." That is why we hear it as one singular note. The reason for this "odd output" is that our input data set sounds very similar to this output. We sample the input in such a way that the data our model trains on is similar sounding.

How to test/ show the discriminator of the MidiNet works well on detecting bad generated songs?

Response: There is no such thing we define as a "bad" song. We only define songs in terms of their complexity. The reason the song may have sounded "bad" was because it was trained on a dataset that sounded like that. We also did not implement MidiNet due to its large complexity. If we were to have used MidiNet then any sound or pattern not found in the training set would be classified as "non optimal" for the output.

Also having comparisons between the performance of the MidiNet and the Neural Composer Model would be nice,

like which model performs better?

Response: We opted out of the MidiNet because the Neural Composer itself was unable to run properly. But our hypothesis is that this is subjective. Given the closed loop nature of GANs, we predict that overtime MidiNet would be the preferred. It is difficult to tell without actually implementing it

Could you please give more details on how to measure the performance of the models? Like how to score the output songs?

Response: A song scored points on complexity based on its ability to produce a number of unique notes and the number of times a note was repeated. This is because the number of times a note repeats itself in a song shows tonality. The frequency at which it repeats shows rhythm. The number of unique notes present also shows the range that the song is produced in. A large number of unique notes would tell us that there is not much learning happening, just a jumble of random notes. Beyond our mathematical analysis none of us are experts in music and therefore we did not know other ways to measure the performance.

8.3. Group 27

Really interesting topic. Thorough literature review which helped provide more background to the topic. Nice connection between the architecture used and that used in the literature review sources. It was nice to see the results of the song produced. Good dataset description, would have been interesting to hear how the dataset used compared to that used by the literature review sources cited. Not necessary for a report, but it would be interesting if you have some commentary on how complexity (and what kinds of complexity) of musical dataset affects model performance, and how musical complexity relates to other dataset complexity we have heard more about in class, like imagery.

Response: Of Course training datasets affect music complexity! The original Neural Composer was trained on game music but ours was trained on a more abstract dataset. That is why when you compare our output to Hack Poet', you will notice that his sounds are more robust while ours sounds more like beeps and pops. Convolutional GAN architecture block diagram slide is nice but it would have been useful to have the pointer move around while talking about it so it's easier to follow. The presentation portion ran a little long. It seems like the model is learning chord progressions, do you think that it will be apparent with more training or will you need to train them separately?

Response: Yes! The sound of the output song is heavily dependent on the training set. So say you gave it classical music, you would hear GAN songs in the classical format that would follow the same structure.

Do you think this is because some of the most common structures in jazz are chord progressions but there are not as many repetitive notes? Any commentary on the result we heard and what you might expect or design towards would be interesting.

Response: Jazz is the easiest for GANs to learn because there are not as many repetitive notes. This allows the output song to be more free form. The output you heard was a composition of multiple notes played on rhythm. It sounds like this because the songs in the data set it sampled sounded like this as well. The model will learn what it is given and output from a subset of the input.

Overall it seems like a really complex problem and a lot of work put into your model creation, very impressive!

9. REFERENCES

[1] Li-Chia Yang, Szu-Yu Chou, Yi-Hsuan Yang. MIDINET: A CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORK FOR SYMBOLIC-DOMAIN MUSIC GENERATION

[2] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, "MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment," in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

[3] Colin Raffel, "Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching," PhD Thesis, 2016.

[4] Carnovalini, F., Rodà, A. (2020, March 9). Computational Creativity and Music Generation Systems: An Introduction to the State of the Art. Frontiers.

[5] HackerPoet. (n.d.). HackerPoet/Composer. GitHub. https://github.com/HackerPoet/Composer.